TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Minimizing flow-time on unrelated machines

Citation for published version (APA):
Bansal, N., & Kulkarni, J. (2014). *Minimizing flow-time on unrelated machines*. (arXiv.org; Vol. 1401.7284 [cs.DS]). s.n.

Document status and date:
Published: 01/01/2014

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 17. Jun. 2021

arXiv:1401.7284v1 [cs.DS] 28 Jan 2014

Minimizing Flow-Time on Unrelated Machines

Nikhil Bansal * Janardhan Kulkarni †

Abstract

We consider some flow-time minimization problems in the unrelated machines setting. In this setting, there is a set of m machines and a set of n jobs, and each job j has a machine dependent processing time of p_{ij} on machine i. The flow-time of a job is the amount of time the job spends in the system (completion time minus its arrival time), and is one of the most natural quality of service measure. We show the following two results: an $O(\min(\log^2 n, \log n \log P))$ approximation algorithm for minimizing the total-flow time, and an $O(\log n)$ approximation for minimizing the maximum flow-time. Here P is the ratio of maximum to minimum job size. These are the first known poly-logarithmic guarantees for both the problems.

*Department of Mathematics and Computer Science, Eindhoven University of Technology, Netherlands. n.bansal@tue.nl.
Supported by the NWO Vidi grant 639.022.211.

†Department of Computer Science, Duke University , 308 Research Drive, Durham, NC 27708. kulkarni@cs.duke.edu.
Supported by NSF Awards CCF-1008065 and IIS-0964560.

1 Introduction

Scheduling a set of jobs over a heterogeneous collection of machines to optimize some quality of service (QoS) measure is one of the central questions in scheduling theory. In modern computing environments be it web-servers, data-centers, clusters of machines or personal computers, heterogeneity of the processors and architectures is ubiquitous. The most general and widely studied model that incorporates the heterogeneity of jobs and machines is the so-called *unrelated machines* setting. Here, there is a set J of n jobs and a set M of m machines. Each job j is specified by its release time (or arrival time) r_j, which is the first time instant it is available for processing, and a (machine dependent) processing requirement p_{ij}, which is the time take to execute j if run on machine i. In this paper, we consider some flow-time related objectives. The flow-time of a job, defined as the amount of time the job spends in the system, is one of most natural measures of quality of a service received by a job. In particular, if a job j completes its processing at time C_j, then flow-time of the job F_j is defined $C_j - r_j$; i.e., its completion time minus its arrival time.

We consider two basic offline problems in the unrelated machines setting: (i) Minimizing the total flow-time of jobs and (ii) Minimizing the maximum flow-time. Both these objectives have been studied extensively in previous works (as we discuss below). However all these results were known only in more restricted settings, and obtaining non-trivial approximations for the general unrelated machines setting was wide open.

More restricted settings: For scheduling on multiple machines, various different models have been studied in the literature. The simplest one is the identical machines setting, where the processing time of a job is identical on all the machines ($p_{ij} = p_j$ for all i). A more general model is the related machines setting, where each machine has a speed s_i and a job has size p_j (that is, $p_{ij} = p_j/s_i$). Another model is the restricted assignment setting, where a job j has a fixed size, but it can only be processed on some subset S_j of the machines (that is, $p_{ij} \in \{p_j, \infty\}$). Clearly, all of these are special cases of the unrelated machines setting.

Minimizing Total flow-time: This objective has been studied extensively in both the offline and online settings, and in various models (see section 1.1 for more references). In the single machine case, it is well known that the SRPT (Shortest Remaining Processing Time) algorithm is optimal for minimizing the total flow-time if preemption is allowed, that is, when a job can be interrupted arbitrarily and resumed later from the point of interruption without any penalty. If preemption is disallowed, the problem becomes much harder and cannot be approximated better than $\Omega(n^{1/2})$ unless P=NP [16]. We will consider only preemptive algorithms in this paper.

For multiple machines, the first breakthrough result was due to Leonardi and Raz [18] who showed that SRPT is an $O(\log(\min(\frac{n}{m}, P)))$ competitive (online) algorithm on identical machines. Here P is the ratio of the maximum to the minimum job size. They also showed that no deterministic online algorithm can do better. Subsequently, other algorithms with similar competitive ratio, but other desirable properties such as no-migration and immediate dispatch were also obtained [5, 18, 4]. Later, poly-logarithmic offline and online guarantees were also obtained for the related machines setting [12, 11]. Interestingly, Garg and Kumar [14] showed that things do not improve much in the offline setting. In particular, no $O(\log^{1-\epsilon} P)$ approximation exists for any $\epsilon > 0$, even for identical machines, unless P=NP.

The above approaches do not seem to help in the restricted assignment case, which is much harder (for example, any online algorithm here must be $\Omega(P)$ competitive [13]). In an important breakthrough, Garg and Kumar [13] obtained an $O(\log P)$ approximation for the problem, based on an elegant and non-trivial LP rounding approach. In particular, they consider a natural LP relaxation of the problem, and round it based on computing certain unpalatable flows [10] on an appropriately defined graph.

In an attempt to extend these ideas to the unrelated machines case, [14] introduce a (α, β)-variability setting (see [14] for details of the model) and prove a general result that in particular implies logarithmic approximations for both restricted assignment and related machines setting. For the unrelated case, their result implies an $O(k)$ approximation where the processing lengths p_{ij} (of all jobs, over all machines) take k different values. This result was also obtained independently by Sitters [20] using matching based techniques (the definition of k [20] is slightly different). In general however k could be as large as nm, and hence a key open question (see e.g. [11, 20] has been whether a poly-logarithmic approximation exists for unrelated machines.

Our first main result gives a positive answer to this question.

Theorem 1.1. *There exists a polynomial time $O(\log n \cdot \log P)$-approximation algorithm for minimizing the sum of flow-times in the unrelated machine setting with preemptions.*

Using a standard trick (see section 2.4) this implies an $O(\log^2 n)$ approximation, which may be better if P is super-polynomial in n.

Our algorithm is based on applying the iterated rounding framework to a new time-indexed linear programming formulation for the problem. The formulation we consider is different from those considered previously, and has much fewer constraints than the natural time-indexed formulation. The fewer constraints are critical in allowing the use of iterated rounding. We describe the new formulation and give an overview of the algorithm in section 2. Theorem 1.1 is proved in section 2.

Maximum flow-time: In many settings, it may be desirable to ensure that *every* job experiences a low delay, instead of guaranteeing low average delay (which is equivalent to total flow-time). Thus a natural objective is to minimize the maximum flow-time over all the jobs. This objective is quite different in flavor from average flow-time. In particular, most algorithms minimizing average flow-time give priority to smaller jobs over longer ones, and hence large jobs tend to suffer unfairly huge delays. Maximum flow-time is in fact closely related to deadline scheduling problems, as the maximum flow-time is D if and only if every job j released at r_j is completed by time $r_j + D$.

For a single machine, First In First Out (FIFO) is an optimal (online) algorithm for minimizing the maximum flow-time. For multiple machines, a 3 competitive online algorithm is known in the identical machine setting [1]. In more general settings, only results based on the resource augmentation analysis are known (see section 1.1 for more details). In particular, no non-trivial approximation algorithm is known for the problem in the unrelated machine setting.

Our second main result is the following.

Theorem 1.2. *There is an $O(\log n)$ polynomial time approximation algorithm for minimizing the maximum flow-time in the unrelated machine setting.*

In fact, the algorithm computes a solution whose maximum flow-time exceeds (additively) the optimum value by at most $O(\log n)p_{\max}$, where $p_{\max}$ is the maximum size of a job in the optimum schedule.

To obtain the result we again write a natural linear programming relaxation of the problem with few constraints, and then apply the iterated rounding technique. Theorem 1.2 is proved in section 3. The key difference here from the proof of theorem 1.1 is that in the final schedule we must ensure that *no* job is delayed by too much.

1.1 Additional Related Work

We briefly survey some additional closely related results. While we only consider unweighted total flow-time, the weighted version has also been extensively studied. For single machine, the best known approxima-

tion is $O(\log \log P)$ [7], and several poly-logarithmic online algorithms are also known [9, 6]. For multiple machines, no $n^{o(1)}$ can exist for total weighted flow-time (via an easy reduction from 3d-matching) and hence all results here are in the so-called resource augmentation setting (see e.g. [8, 3]). A comprehensive survey of various flow-time related results can be found in [15, 19].

Recently, maximum flow-time was studied by Anand et al. [2] in the online setting. They showed a lower bound of $\Omega(m, \text{poly}(n))$ for the restricted assignment case, and gave an algorithm for the unrelated machines case in the resource augmentation model.

2 Minimizing the total flow-time

In this section, we consider the problem of minimizing total flow-time on unrelated machines and prove theorem 1.1.

2.1 Alternate LP relaxation and the high-level idea

Standard LP formulation: Before describing the new LP formulation that we use, we first describe the standard time indexed linear programming relaxation of the problem that was used for example in [13, 14]. There is a variable x_{ijt} for each machine $i \in [m]$, each job $j \in [n]$ and each unit time slot $t \geq r_j$. The x_{ijt} variables indicate the amount to which a job j is processed on machine i during the time slot t. The first set of constraints (1) says that every job must be completely processed. The second set of constraints (2) says that a machine cannot process more than one unit of job during any time slot. Note that this LP allows a job to be processed a job on multiple machines, and even at the same time.

$$\begin{aligned}
\min \sum_{i,j,t} \left(\frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) \cdot x_{ijt} & \\
\text{s.t.} \quad \sum_{i} \sum_{t \geq r_j} \frac{x_{ijt}}{p_{ij}} \geq 1 & \qquad \forall j \qquad (1) \\
\sum_{j \,:\, t \geq r_j} x_{ijt} \leq 1 & \qquad \forall i, t \qquad (2) \\
x_{ijt} \geq 0 & \qquad \forall i, j, t \geq 0
\end{aligned}$$

Fractional flow-time: The objective function needs some explanation. The term $\sum_{i,t} x_{ijt}$ is precisely the total amount of processing done on job j. The term $\sum_{i,t}(t - r_j) \cdot \frac{x_{ijt}}{p_{ij}}$ is the *fractional* flow-time of job j and we denote it by f_j. The (integral) flow-time of a job j can be viewed as summing up 1 during each time step that j is alive, i.e. $\sum_{t \geq r_j : j \text{ is alive at } t} 1$. The fractional flow-time instead is the sum over time instants of the remaining (unfinished) fraction of job j. Note that on machine i, the fraction of job j that is unfinished at time t is $\sum_{t' > t} \frac{x_{ijt}}{p_{ij}}$ (the numerator denotes the amount of work that will be done j on machine i at time after t). Thus the fractional flow-time on machine i is $\sum_{t \geq r_j} \sum_{t' > t} \frac{x_{ijt}}{p_{ij}}$, which by collecting the x_{ijt} terms is exactly equal to $\sum_{i,t}(t - r_j) \cdot \frac{x_{ijt}}{p_{ij}}$. It can be easily checked that the integral flow-time is at least the fractional flow-time plus half the size of a job. Thus the objective function in the above LP is valid lowerbound on the optimal solution. For more details on the fractional flow-time and the LP above, see [13].

We assume that $\min_{i,j} p_{ij} \neq 0$ (if $p_{ij} = 0$, we simply schedule j on machine i upon arrival). Define $P = \max_{i,j} p_{ij} / \min_{i,j} p_{ij}$. Without loss of generality, we assume henceforth that $\min_{i,j} p_{ij} = 1$. For

$k = 0, 1, \ldots, \log P$, we say that a job j belongs to class k on machine i if $p_{ij} \in (2^{k-1}, 2^k]$. Note that the class of a job depends on the machine. We now give a new LP relaxation for the problem. The main idea is that we do not enforce the capacity constraints (2) for each time slot, but instead only enforce these constraints over carefully chosen intervals of time. The advantage of this relaxation is that it has relatively few constraints, which is will be useful in applying the iterated rounding approach to it. Even though the number of constraints is fewer, as we will see the quality of the relaxation is not sacrificed much.

New LP formulation: In the new LP relaxation of the problem there is a variable y_{ijt} (similar to x_{ijt} before) that denotes the total units of job j processed on machine i at time t. However, unlike the time indexed relaxation, we allow y_{ijt} to take values greater than one. In fact, we will round the new LP in such a way that eventually $y_{ijt} = p_{ij}$ for each job, which will have a natural interpretation that job j is scheduled at time t on machine i.

For each class k and each machine i, we partition the time horizon $[0, T]$ into intervals of size $4 \cdot 2^k$. Without loss of generality we can assume that $T \leq nP$ (otherwise the input instance can be trivially split into two disjoint non-overlapping instances). For $a = 1, 2, \ldots$, let $I(i, a, k)$ denote the a-th interval of class k on machine i. That is, $I(i, 1, k)$ is the time interval $[0, 4 \cdot 2^k]$ and $I(i, a, k) = ((4 \cdot 2^k)(a-1), (4 \cdot 2^k)a]$.

We write the new LP relaxation.

$$\sum_i \sum_{t \geq r_j} \sum_k \sum_{j \in (2^{k-1}, 2^k]} \left(\frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) \cdot y_{ijt} \qquad (\mathsf{LP}_{\mathsf{new}})$$

$$\text{s.t.} \quad \sum_i \sum_{t \geq r_j} \frac{y_{ijt}}{p_{ij}} \geq 1 \qquad \forall j \qquad (3)$$

$$\sum_{j\,:\,p_{ij} \leq 2^k} \sum_{t \in I(i,a,k)} y_{ijt} \leq \text{Size}(I(i, a, k)) \qquad \forall i, k, a \qquad (4)$$

$$y_{ijt} \geq 0 \qquad \forall i, j, t \,:\, t \geq r_j$$

Here, $\text{Size}(I(i, a, k))$ denotes the size of the interval $I(i, a, k)$ which is $4 \cdot 2^k$ (but would change in later iterations of the LP when we apply iterated rounding). Observe that in (4) only jobs of class $\leq k$ contribute to the left hand side of constraints corresponding to intervals of class k.

Clearly, ($\mathsf{LP}_{\mathsf{new}}$) is a relaxation of the time indexed LP formulation considered above, as any valid solution there is also a valid solution to ($\mathsf{LP}_{\mathsf{new}}$) (by setting $y_{ijt} = x_{ijt}$). Therefore, we conclude that an optimum solution to ($\mathsf{LP}_{\mathsf{new}}$) lower bounds the value optimal solution.

Remark: When we apply iterative rounding and consider subsequent rounds, we will refer the intervals $I(i, a, k)$ as $I(i, a, k, 0)$ (to indicate that they are intervals from the 0-th round). However we drop 0 from description above for now.

The high-level approach: The main idea of our algorithm is the following. Let us call a job j to be *integrally assigned* to machine i at time t, if $y_{ijt} = p_{ij}$ (note that this job will be completely executed on machine i). Let us view this as processing the job j during $[t, t + p_{ij})$. In the algorithm, we first find a *tentative* integral assignment of jobs to machines (at certain times) such that the total flow-time of this solution is at most the LP value. This solution is tentative in the sense that multiple jobs could use the same time slot; however we will ensure that the effect of this overlap is negligible (in the sense of Lemma 2.1 below).

More precisely, we show the following result.

Lemma 2.1. *There exists a solution $y^* = \{y^*_{ijt}\}_{i,j,t}$ satisfying the following properties.*

- *(*Integrality:*) For each job j, there is exactly one non-zero variable y_{ijt} in y^*, which takes value p_{ij}. That is, each job is assigned integrally to exactly one machine, and one time slot : $y^*_{ijt} = p_{ij}$.*
- *(*Low cost:*) The cost of y^* is at most the cost of an optimal solution to $\mathsf{LP}_{\mathsf{new}}$.*
- *(*Low overload:*) For any interval of time $[t_1, t_2]$, every machine i and for every class k,*

$$\sum_{j\,:\,p_{ij}\le 2^k} \sum_{t\in[t_1,t_2]} y^*_{ijt} \le (t_2 - t_1) + O(\log n)\cdot 2^k.$$

  That is, the total size of jobs of class at most k assigned integrally in any time interval $[t_1, t_2]$ exceeds the size of the interval by at most $O(\log n)\cdot 2^k$.

Lemma 2.1 is the core of our algorithm, which will be proved using iterated rounding. In particular, we show using a counting argument that in each round a basic feasible optimum solution assigns at least a constant fraction of jobs integrally in each round. Therefore, after $O(\log n)$ rounds every job is integrally assigned to some machine. In each round as some jobs get integrally assigned, we will fix them permanently and reduce the free space available in those intervals. Then, we merge these intervals greedily to ensure that the free space in an interval corresponding to class k stays $O(1)\cdot 2^k$. This merging process adds an overload of at most $O(1)\cdot 2^k$ to any time interval in each round. This ensures that the total error added for any time interval is $O(\log n)\cdot 2^k$.

The next step is to show that the tentative schedule can be converted to a valid preemptive schedule by increasing the total flow-time of jobs by $O(\log P \log n)$ times the $\mathsf{LP}_{\mathsf{new}}$ value. To this end, we use ideas similar to those used by [11, 13] for the related or restricted machines case. In particular, we schedule the jobs on each machine in the order given by the tentative schedule, while prioritizing the jobs in the shortest job first (SJF) order. The low overload property of the tentative schedule ensures that a job of class k is additionally delayed by at most $O(\log n)\cdot 2^k$ due to jobs that arrive before it, or is delayed by smaller jobs (of strictly lower class) that arrive after the time when it is tentatively scheduled. In either case, we show that this delay can be charged to the total flow-time of other jobs.

We now give the details. We first describe in section 2.2 how to convert a tentative schedule y^* satisfying the conditions of Lemma 2.1 to a proper one, and show how this implies Theorem 1.1. In section 2.3 we describe the iterated rounding algorithm and prove lemma 2.1.

2.2 Tentative Schedule to Actual Schedule

We show how Theorem 1.1 follows given a solution y^* satisfying the conditions of Lemma 2.1. Recall that in the solution y^*, for each job j, we have $y_{ijt} = p_{ij}$ for some time instant t and some machine i, but this is not necessarily a valid schedule. We convert y^* into a valid preemptive schedule S as follows. Fix a machine i and let $J(i, y^*)$ denote the set of jobs which are scheduled on machine i in the solution y^* (i.e. jobs j such that $y_{ijt} = p_{ij}$ for some time instant t). In the schedule S, for each machine i, we imagine that a job j in $J(i, y^*)$ becomes available for S at the time t where $y_{ijt} = p_{ij}$. We schedule the jobs in S (after they become available) using Shortest Job First (SJF) (where jobs in the same class are viewed as having the same size); for two jobs belonging to same class we schedule the jobs in the order given by y^*. Let $J_k(i, S)$ denote the set of jobs of class k which are assigned to machine i in schedule S, and let $J(i, S) = \cup_k J_k(i, S)$ denote the set of jobs scheduled by S on i. Clearly, $J_k(i, S) = J_k(i, y^*)$. We also observe that, since jobs within a class are considered in order, for each class k and on each machine i, there is at most one job belonging to

class k which is partially processed (due to preemptions by jobs of a smaller class). This directly implies the following relation between the fractional and integral flow-time of jobs in S. Let F_j^S denote the flow-time of job j in schedule S and f_j^S denote the fractional flow-time.

Lemma 2.2. *Fix a machine i and the set of jobs belonging to class k. Then,*

$$\sum_{j \in J_k(i,S)} F_j^S \leq \sum_{j \in J_k(i,S)} f_j^S + \sum_{j \in J(i,S)} p_{ij}.$$

Remark: Note that first two summations are over $J_k(i,S)$, while the third summation is over $J(i,S)$.

Proof. We use the alternate view of integral and fractional flow-times. Let C_j^S denote the completion time of job j in the schedule S. Then, the integral flow-time of j is $F_j^S = \int_{t=r_j}^{C_j^S} 1 \cdot dt$ and the fractional flow-time is $f_j^S = \int_{t=r_j}^{C_j^S} p_{ij}(t)/p_{ij} dt$, where $p_{ij}(t)$ denotes the remaining processing time of job j on machine i.

Let $J_k(i,S,t)$ denote the set of jobs available for processing at time t of class k on machine i in S, which have not been completed, and $\mathcal{T}(i,k)$ denote the set of time instants where $J_k(i,S,t) \geq 1$, i.e. at least one job of class k is alive. Then,

$$\sum_{j \in J_k(i,S)} F_j^S = \int_{t \in \mathcal{T}(i,k)} |J_k(i,S,t)| dt \leq \int_{t \in \mathcal{T}(i,k)} \left(1 + \sum_{j \in J_k(i,S)} \frac{p_{ij}(t)}{p_{ij}}\right) dt \leq \sum_{j \in J(i,S)} p_{ij} + \sum_{j \in J_k(i,S)} f_j^S.$$

The first inequality follows as there is at most one partially processed job of class k at any time in S. The second inequality follows by observing that $\int_{t \in \mathcal{T}(i,k)} 1 dt$ is simply the time units when at least one class k job is alive. This can be at most the time when any job (of any class) is alive, which is precisely equal to $\sum_{j \in J(i,S)} p_{ij}$, the total processing done on machine i (as the schedule S is never idle if there is work to be done). Thus, $\int_{t \in \mathcal{T}(i,k)} 1 dt \leq \sum_{j \in J(i,S)} p_{ij}$. Moreover, $\int_{t \in \mathcal{T}(i,k)} \sum_{j \in J_k(i,S)} \frac{p_{ij}(t)}{p_{ij}} dt = \sum_{j \in J_k(i,S)} \int_{t \geq r_j} \frac{p_{ij}(t)}{p_{ij}} dt$ which is exactly the total fractional flow-time $\sum_{j \in J_k(i,S)} f_j^S$. □

Let $V_k(y^*, i, t)$ denote the total remaining processing time (or volume) of jobs of class k alive at time t on machine i in the schedule defined by y^* (i.e. these are precisely the jobs that are released but not yet scheduled by t); similarly, let $V_k(S, i, t)$ denote the total remaining processing time of jobs of class k that have $r_j \leq t$, but are unfinished at time t on machine i in the schedule S. As a job is available for S only after it is scheduled in y^*, we make the following simple observation.

Observation 1. *For any k, $V_k(y^*, i, t) \leq V_k(S, i, t)$. Moreover,$V_k(S, i, t) - V_k(y^*, i, t)$ is the volume of precisely those jobs of class k that are available to S (i.e. already scheduled in y^*), but have not been completed by S.*

We now show the crucial lemma that $V_k(y^*, i, t)$ and $V_k(S, i, t)$ do not deviate by too much.

Lemma 2.3. *For machine i and class k, $\forall t, V_k(S, i, t) - V_k(y^*, i, t) \leq O(\log n) \cdot 2^k$*

Proof. By Observation 1, $V_k(S, i, t) - V_k(y^*, i, t)$ is the total processing time of jobs of class k that are available for processing in S at time t and not yet completed. As $V_k(S, i, t) - V_k(y^*, i, t) \leq V_{\leq k}(S, i, t) - V_{\leq k}(y^*, i, t)$ (this follows by Observation 1 as $V_{k'}(S, i, t) \geq V_{k'}(y^*, i, t)$ for each k'), it suffices to bound the

latter difference. Let $t' \leq t$ be the last time before t when machine i was idle in S, or was processing a job of class strictly greater than k. This means that no jobs of class $\leq k$ are available to S (as they have either not arrived or have not yet been made available by y^*). Thus, $V_{\leq k}(S,i,t') = V_{\leq k}(y^*,i,t')$ or equivalently $V_{\leq k}(S,i,t') - V_{\leq k}(y^*,i,t') = 0$. By the low overload property, the total volume of jobs belonging to class at most k that becomes available during $(t',t]$ is at most $(t-t') + O(\log n)2^k$. Since S processes only jobs of class at most k during $(t',t]$ (by definition of t'), S completes precisely $t-t'$ volume of jobs belonging to class at most k. This implies that $V_{\leq k}(S,i,t) - V_{\leq k}(y^*,i,t) = O(\log n)2^k$. □

We are now ready to show how this implies Theorem 1.1

Proof of Theorem 1.1. We first compare the fractional flow-times of schedules defined by y^* and S and then use Lemma 2.2 to complete the argument.

Define y^S_{ijt} variables corresponding to the schedule S by setting y^S_{ijt} to amount of processing done on job j on machine i at time t in the schedule S . Let $P(S,i) = \sum_{j \in J(i,S)} \sum_t y^S_{ijt}$ denote the total processing time of the jobs scheduled on machine i in S. Clearly, since the set of jobs on machine i in y^* and S is identical, we have $P(S,i) = P(y^*,i)$. Let $\mathcal{T}(i,k)$ be the times when there is at least one available but unfinished job in S. Recall that $\int_{t \in \mathcal{T}(i,k)} 1 \cdot dt = P(i,S)$.

Then, the difference between the fractional flow-times of jobs in S and y^* can be bounded by

$$\begin{aligned}
\sum_j (f_j^S - f_j^{y^*}) &= \sum_i \sum_t \sum_k \sum_{j: p_{ij} \in (2^{k-1}, 2^k]} (y^S_{ijt} - y^*_{ijt}) \cdot \left(\frac{t - r_j}{p_{ij}} \right) \\
&\leq \sum_i \sum_t \sum_k \sum_{j: p_{ij} \in (2^{k-1}, 2^k]} (y^S_{ijt} - y^*_{ijt}) \cdot \left(\frac{t - r_j}{2^{k-1}} \right) \\
&= \sum_i \sum_t \sum_k \sum_{j: p_{ij} \in (2^{k-1}, 2^k]} \frac{1}{2^{k-1}} (V_k(S,i,t) - V_k(y^*,i,t)) \qquad (5) \\
&\leq \sum_i \sum_k \sum_{t \in \mathcal{T}(i,k)} O(\log n) \qquad \text{[By Lemma (2.3)]} \\
&= \sum_i \sum_k O(\log n) P(i,S) \qquad (6) \\
&\leq \sum_i O(\log n \cdot \log P) P(i,S) \\
&= O(\log n \cdot \log P) P(S)
\end{aligned}$$

Here (5) follows as for any schedule S, the quantity $\sum_{j: p_{ij} \in (2^{k-1}, 2^k]} \sum_{t \geq r_j} y^S_{ijt}(t - r_j)$ is exactly equal to $\sum_t V_k(S,i,t)$ (by the two different ways of looking at fractional flow-time).

Now we can bound the total flow-time as

$$\begin{aligned}
\sum_j F_j^S &= \sum_i \sum_k \sum_{j \in J_k(i,S)} F_j^S \\
&\leq \sum_i \sum_k \left(\sum_{j \in J_k(i,S)} f_j^S + \sum_{j \in J(i,S)} p_{ij} \right) \qquad [\text{ By Lemma (2.2)}] \\
&= \sum_j f_j^S + \sum_i \sum_k \sum_{j \in J(i,S)} p_{ij} \\
&\leq \sum_j f_j^S + O(\log P) P(S) \\
&\leq \sum_j f_j^{y^*} + O(\log n \cdot \log P) P(S)
\end{aligned}$$

which is at most $O(\log n \cdot \log P)$ times the value of optimal solution to $\mathsf{LP}_{\mathsf{new}}$. □

2.3 Iterated Rounding of $\mathsf{LP}_{\mathsf{new}}$ and proof of Lemma 2.1

In this section we prove the Lemma 2.1 using iterated rounding. In the iterated rounding technique, we successively relax the $\mathsf{LP}_{\mathsf{new}}$ with a sequence of linear programs, each having fewer constraints than the previous one while ensuring that optimal solutions to the linear programs is at most the cost of optimal solution to $\mathsf{LP}_{\mathsf{new}}$. An excellent reference for various applications of this technique is [17].

We denote the successive relaxations of $\mathsf{LP}_{\mathsf{new}}$ by $LP(\ell)$ for $\ell = 0, 1, \ldots$. Let $J(\ell)$ denote the set of jobs that appear in $LP(\ell)$. Linear program $LP(0)$ is same as $\mathsf{LP}_{\mathsf{new}}$, and $J(0) = J$. We define $LP(\ell)$ for $\ell > 0$ inductively as follows.

- *Computing a basic optimal solution:* Find a basic optimal solution $y^*(\ell-1) = \{y_{ijt}^{\ell-1}\}_{i,j,t}$ to $LP(\ell-1)$. We use $y_{ijt}^{\ell-1}$ to indicate the value taken by the variable y_{ijt} in the solution $y^*(\ell-1)$. Let $\mathcal{S}_{\ell-1}$ be the set of variables in the support of $y^*(\ell-1)$. We initialize $J(\ell) = J(\ell-1)$.

- *Eliminating 0-variables:* The variables y_{ijt} for $LP(\ell)$ are defined only for the variables in $\mathcal{S}_{\ell-1}$. That is, if $y_{ijt}^{\ell-1} = 0$ in $y^*(\ell-1)$, then these variables are fixed to 0 forever, and do not appear in $LP(\ell)$.

- *Fixing integral assignments:* If a variable $y_{ijt}^{\ell-1} = p_{ij}$ in $y^*(\ell-1)$ for some job j, then j is permanently assigned to machine i at time t in y^* (as required by Lemma 2.1), and we update $J(\ell) = J(\ell) \setminus \{j\}$. We drop all the variables corresponding to the job j in $LP(\ell)$, and also drop the service constraint (8) for the job j. We use $A(\ell-1)$ to denote the set of jobs which get integrally assigned in $(\ell-1)$-th iteration. We redefine the intervals based on the unassigned jobs next.

  Remark: It will be convenient below not to view an interval as being defined by its start and end times, but by the y_{ijt}-variables it contains.

- *Defining intervals for ℓ-th iteration:* Fix a class k and machine i. We define the new intervals $I(i, *, k, \ell)$ and their sizes as follows.

  Consider the jobs in $J(\ell)$ (those not yet integrally assigned) belonging to classes $\leq k$, and order the variables y_{ijt} in increasing order of t (in case of ties, order them lexicographically). Greedily

group consecutive y_{ijt} variables (starting from the beginning) such that sum of the $y_{ijt}^{\ell-1}$ values of the variables in that group first exceeds $4 \cdot 2^k$.

Each such group will be an interval (which we view as a subset of y_{ijt} variables). Define the size of an interval $I = I(i, *, k, \ell)$ as

$$\text{Size}(I) = \sum_{y_{ijt} \in I} y_{ijt}^{\ell-1}. \tag{7}$$

As $y_{ijt}^{\ell-1} \leq 2^k$ for jobs of class k, clearly $\text{Size}(I) \in [4 \cdot 2^k, 5 \cdot 2^k]$ for each I (except possibly the last, in which case we can add a couple of extra dummy jobs at the end) .

Note that the intervals formed in $LP(\ell)$ for $\ell > 0$ are not related to time anymore (unlike $LP(0)$), and in particular can span much longer duration of time that $4 \cdot 2^k$. All we ensure is that the amount of unassigned volume in an interval is $\Omega(2^k)$.

Defining the LP for ℓ-th iteration: With the above definition intervals $I(i, a, k, \ell)$ and the y_{ijt} variables defined for the ℓ-th iteration, we write the linear programming relaxation for ℓ-th round, $LP(\ell)$.

$$\sum_i \sum_{t \geq r_j} \sum_k \sum_{j \in J(\ell): j \in (2^{k-1}, 2^k]} \left(\frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) \cdot y_{ijt} \tag{LP(ℓ)}$$

$$\text{s.t.} \quad \sum_i \sum_{t \geq r_j} \frac{y_{ijt}}{p_{ij}} \geq 1 \qquad \forall j \in J(\ell) \tag{8}$$

$$\sum_{y_{ijt} \in I(i,a,k,\ell)} y_{ijt} \leq \text{Size}(I(i, a, k, \ell)) \qquad \forall i, k, a \tag{9}$$

$$y_{ijt} \geq 0 \qquad \forall i, j \in J(\ell), t : t \geq r_j$$

2.3.1 Analysis

We note that $LP(\ell)$ is clearly a relaxation of $LP(\ell - 1)$ (restricted to variables corresponding to jobs in $J(\ell)$). This follows as setting $y_{ijt} = y_{ijt}^{\ell-1}$ is a feasible solution for $LP(\ell)$ (by the definition of $\text{Size}(I)$). Moreover, the objective function of $LP(\ell)$ is exactly the objective of $LP(\ell - 1)$ restricted to variables in $J(\ell)$. Let y^* denote the final integral assignment (assuming it exists) obtained by applying the algorithm iteratively to $LP(0), LP(1), \ldots$. Then this implies that,

Lemma 2.4. *The cost of the integral assignment $cost(y^*)$ is at most the cost of optimal solution to* $\mathsf{LP}_{\mathsf{new}}$.

Bounding the number of iterations: We now show that the sequence of $LP(\ell)$ relaxations terminate after some small number of rounds. Let $N_\ell = |J(\ell)|$ denote the number of jobs in $LP(\ell)$ (i.e. the one unassigned after solving $LP(\ell - 1)$).

Lemma 2.5. *After each iteration, the number of unassigned jobs decreases by a constant factor. In particular, for each ℓ: $N_\ell \leq N_{\ell-1}/2$.*

Proof. Consider the basic optimal solution $y^*(\ell-1)$ to $LP(\ell-1)$. Let $\mathcal{S}_{\ell-1}$ denote the non-zero variables in this solution, i.e. $y^{\ell-1}_{ijt}$ such that $y^{\ell-1}_{ijt} > 0$. Consider a linearly independent family of tight constraints in $LP(\ell-1)$ that generate the solution $y^*(\ell-1)$. As tight constraints $y^{\ell-1}_{ijt} = 0$ only lead to 0 variables, it follows that $|\mathcal{S}_{\ell-1}|$ is at most the number of tight constraints (8) or tight capacity constraints (9). Let $C_{\ell-1}$ denote the number of tight capacity constraints. Thus,

$$|\mathcal{S}_{\ell-1}| \leq N_{\ell-1} + C_{\ell-1}. \tag{10}$$

Recall that $A(\ell-1)$ denotes the set of jobs that are assigned integrally in the solution $y^*(\ell-1)$. As each job not in $A(\ell-1)$ contributes at least two to $|\mathcal{S}_{\ell-1}|$, we also have

$$|\mathcal{S}_{\ell-1}| \geq |A(\ell-1)| + 2(N_{\ell-1} - |A(\ell-1)|) = N_{\ell-1} + N_\ell. \tag{11}$$

The equality above follows as $N_\ell = N_{\ell-1} - |A(\ell-1)|$ is the number of the (remaining) jobs considered in $LP(\ell)$. Together with (10) this gives

$$N_\ell \leq C_{\ell-1}. \tag{12}$$

We now show that $C_{\ell-1} \leq N_{\ell-1}/2$, which together with (12) would imply the claimed result. We do this by a charging scheme. Assign two tokens to each job j in $N_{\ell-1}$. The jobs redistribute their tokens as follows.

Fix a job j and let $k(i)$ denote the class of j on machine i. For each machine i, time t and class $k' \geq k(i)$, the job j gives $\frac{1}{2^{k'-k(i)}} \frac{y^{\ell-1}_{ijt}}{p_{ij}}$ tokens to the class k' interval $I(i,a,k',\ell-1)$ on machine i containing y_{ijt}. If there are multiple time slots t in an interval $I(i,a,k',\ell-1)$ with $y^{\ell-1}_{ijt} > 0$, then $I(i,a,k',\ell-1)$ receives a contribution from each of these slots. This is a valid token distribution scheme as the total tokens distributed by the job j is at most

$$\sum_i \sum_t \sum_{k' \geq k(i)} \frac{y^{\ell-1}_{ijt}}{2^{k'-k(i)} \cdot p_{ij}} = \sum_i \sum_t \left(\frac{y^{\ell-1}_{ijt}}{p_{ij}} \cdot \sum_{k' \geq k(i)} \frac{1}{2^{k'-k(i)}} \right) \leq 2 \cdot \sum_i \sum_t \frac{y^{\ell-1}_{ijt}}{p_{ij}} = 2.$$

Next, we show that each tight constraint of type (9) receives at least 4 tokens. If an interval $I(i,a,k',\ell-1)$ of class k' on machine i is tight, this means that $\sum_{y_{ijt} \in I(i,a,k',\ell-1)} y^{\ell-1}_{ijt} = \text{Size}(I(i,a,k',\ell-1))$ which is at least $4 \cdot 2^{k'}$. Now, the tokens given by variable a y_{ijt} in $I(i,a,k',\ell-1)$ where j is of class $k(i) \leq k'$ are

$$\frac{y^{\ell-1}_{ijt}}{(2^{k'-k(i)} \cdot p_{ij})} \geq \frac{y^{\ell-1}_{ijt}}{(2^{k'-k(i)} \cdot 2^{k(i)})} = \frac{y^{\ell-1}_{ijt}}{2^{k'}}.$$

Thus, the tokens obtained by $I(i,a,k',\ell-1)$ are at least $\sum_{y_{ijt} \in I(i,a,k',\ell-1)} y^{\ell-1}_{ijt}/2^{k'} \geq 4 \cdot 2^{k'}/2^{k'} = 4$. As each job distributes at most 2 tokens and each tight interval receives at least 4 tokens, we conclude that $C_{\ell-1} \leq N_{\ell-1}/2$. □

Bounding the Backlog: To complete the proof of Lemma 2.1, it remains to show that for any time period $[t_1, t_2]$ and for any class k, the total volume of jobs belonging to class at most k assigned to $[t_1, t_2]$ in y^* is at most $t_2 - t_1 + O(\log n)2^k$. Recall that $A(\ell)$ denotes the set of jobs which get integrally assigned in the ℓ-th round. We use $A(t_1, t_2, i, k, \ell)$ to denote the set of jobs of class $\leq k$ which get integrally assigned to the machine i in the interval $[t_1, t_2]$.

Given the solution $y^*(\ell)$ to $\mathsf{LP}(\ell)$ and a time interval $[t_1, t_2]$, let us define

$$\mathsf{Vol}(t_1,t_2,i,k,\ell) := \sum_{j\in J(\ell):p_{ij}\le 2^k}\sum_{t\in[t_1,t_2]} y^{\ell}_{ijt} + \sum_{\ell'\le(\ell-1)}\sum_{j\in A(t_1,t_2,i,k,\ell')} p_{ij}$$

as the total size of jobs of class $\le k$, assigned either integrally or fractionally to the period $[t_1,t_2]$ after ℓ rounds. The following key lemma controls how much Vol can get worse in each round.

Lemma 2.6. *For any period $[t_1,t_2]$, machine i, class k and round ℓ,*

$$\mathsf{Vol}(t_1,t_2,i,k,\ell) \le O(1)\cdot 2^k + \mathsf{Vol}(t_1,t_2,i,k,\ell-1).$$

Proof. By the definition of Vol this is equivalent to showing that

$$\sum_{j\in J(\ell):p_{ij}\le 2^k}\sum_{t\in[t_1,t_2]} y^{\ell}_{ijt} + \sum_{j\in A(t_1,t_2,i,k,\ell-1)} p_{ij} \le O(1)\cdot 2^k + \sum_{j\in J(\ell-1):p_{ij}\le 2^k}\sum_{t\in[t_1,t_2]} y^{\ell-1}_{ijt} \tag{13}$$

Fix a time period $[t_1,t_2]$. The main idea is that in each round ℓ, the error to Vol can be introduced only due to the two class k intervals overlapping with the boundary of $[t_1,t_2]$.

Consider the maximal set of contiguous intervals $I(i,b,k,\ell), I(i,b+1,k,\ell),\dots I(i,b+h,k,\ell)$, for some $b,h\ge 0$, that contain the period $[t_1,t_2]$. More precisely, b is the smallest index such that $I(i,b,k,\ell)$ contains some y_{ijt} with $t\in[t_1,t_2]$, and h is the largest index such that $I(i,b+h,k,\ell)$ contains some y_{ijt} with $t\in[t_1,t_2]$. As these intervals have size at most $5\cdot 2^k$, we have

$$\sum_{y_{ijt}\in I(i,b,k,\ell)} y^{\ell}_{ijt} + \sum_{y_{ijt}\in I(i,b+h,k,\ell)} y^{\ell}_{ijt} \le 10\cdot 2^k. \tag{14}$$

Now, consider the intervals $I(i,b',k,\ell)\in\{I(i,b+1,k,\ell), I(i,b+2,k,\ell),\dots I(i,b+h-1,k,\ell)\}$ that are completely contained in $[t_1,t_2]$ (i.e. for all $y_{ijt}\in I(i,b',k,\ell)$, $t\in[t_1,t_2]$).By definition of these intervals and capacity constraints of $\mathsf{LP}(\ell)$ we have,

$$\begin{aligned}
\sum_{b'=b+1}^{b+h-1}\sum_{y_{ijt}\in I(i,b',k,\ell)} y^{\ell}_{ijt} &\le \sum_{b'=b+1}^{b+h-1} \mathrm{Size}(I(i,b',k,\ell)) && \text{[By the constraints (9) of } \mathsf{LP}(\ell)\text{]}\\
&\le \sum_{b'=b+1}^{b+h-1}\sum_{y_{ijt}\in I(i,b',k,\ell)} y^{\ell-1}_{ijt} && \text{[By definition (7) of Size]}\\
&\le \sum_{j\in J(\ell):p_{ij}\le 2^k}\sum_{t\in[t_1,t_2]} y^{\ell-1}_{ijt} && (15)
\end{aligned}$$

We now prove (13). Consider,

$$\begin{aligned}
\sum_{j\in J(\ell):p_{ij}\le 2^k}\sum_{t\in[t_1,t_2]} y^{\ell}_{ijt} &\le \sum_{b'=b}^{b+h}\sum_{y_{ijt}\in I(i,b',k,\ell)} y^{\ell}_{ijt}\\
&\le 10\cdot 2^k + \sum_{j\in J(\ell):p_{ij}\le 2^k}\sum_{t\in[t_1,t_2]} y^{\ell-1}_{ijt} \quad \text{[by (14) and (15)]}\\
&\le 10\cdot 2^k + \sum_{j\in J(\ell-1):p_{ij}\le 2^k}\sum_{t\in[t_1,t_2]} y^{\ell-1}_{ijt} - \sum_{j\in A(t_1,t_2,i,k,\ell-1)} p_{ij}
\end{aligned}$$

The last step follows as $J(\ell) = J(\ell-1) \setminus A(\ell-1)$ and as $\sum_{j \in A(t_1,t_2,i,k,\ell-1)} y^{\ell-1}_{ijt} = \sum_{j \in A(t_1,t_2,i,k,\ell-1)} p_{ij}$. □

This directly implies the following bound on the total error in any period $[t_1, t_2]$ in y^*.

Lemma 2.7. *For a given time period $[t_1, t_2]$, machine i and class k, the total volume of jobs of class at most k, assigned to the interval is at most $(t_2 - t_1) + O(\log n)2^k$.*

Proof. Recall the definition of an interval $I(i, a, k, 0)$ in $LP(0)$. Each interval $I(i, a, k, 0) = (t', t'']$ has size $4 \cdot 2^k$ and contains all the y_{ijt} variables for jobs of class at most k and $t \in (t', t'']$. Therefore, for any period $[t_1, t_2]$, by considering the capacity constraints (4) of $LP(0)$ for the overlapping intervals $I(i, *, k, 0)$, we obtain

$$\mathsf{Vol}(t_1, t_2, i, k, 0) = \sum_{j: p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y^0_{ijt} \leq (t_2 - t_1) + O(1) \cdot 2^k \tag{16}$$

Applying lemma 2.6 inductively (for the term Vol in the above equation) over the $O(\log n)$ iterations of the algorithm gives the result. □

Proof of Lemma 2.1. Consider the final solution y^* at the end of the algorithm. By our construction, each job is integrally assigned in y^*. By Lemma (2.4), cost(y^*) is no more than the cost of an optimal solution to $\mathsf{LP}_{\mathsf{new}}$. By Lemma (2.7), for any time period $[t_1, t_2]$, machine i and class k, the total volume of jobs assigned of jobs in class $\leq k$ is at most $(t_2 - t_1) + O(\log n)2^k$. This concludes the proof. □

2.4 The $O(\log^2 n)$ approximation

The $O(\log^2 n)$ approximation follows directly by observing that jobs much small $p_{\max}$ essentially have no effect.

The algorithm guesses $p_{\max}$, the value of the maximum job size in an optimal solution (say, by trying out all possible mn choices), and considers a modified instance J' where we set $p_{ij} = p_{\max}/n^2$ whenever $p_{ij} < p_{\max}/n^2$, and applies the previous algorithm for J'. Clearly, $P \leq n^2$ for J'. Moreover OPT$(J') \leq$ 2 OPT(J). Indeed, consider the optimum solution for J and for each job j assigned to machine i with size $p_{ij} < p_{\max}/n^2$, increase its size to $p_{\max}/n^2$ and push all the jobs behind it by the amount by which the size increases. This gives a valid schedule for J'. Each job can be pushed by at most n jobs, and hence its flow time increases by at most $n \cdot p_{\max}/n^2$. Thus the total flow-time increases by at most $p_{\max}$ which is at most OPT(J).

3 Minimizing the Maximum flow-time

We now consider the problem of minimizing the maximum flow-time. By doing binary search, we assume that we know the value of optimum solution (OPT), say OPT = D. Let us index the jobs by their release times (breaking ties arbitrarily).

We now write a linear programming relaxation for the problem. In this relaxation, there is a variable x_{ij} denoting the total processing done on job j on machine i. If $p_{ij} > D$ for a job j on machine i, then we set $x_{ij} = 0$, as j cannot be scheduled on machine i. The first set of constraints (17) ensure that each job is completely processed. To see the second constraint (18), we note that any job released during the interval

$[t, t']$ must be completed by time $t' + D$. Thus the total size of jobs released in $[t, t']$ that are assigned to i can be at most $(t' - t) + D$. Moreover, it suffices to consider intervals such that t, t' are release dates of some jobs (as this gives the tightest constraints).

$$\sum_i \frac{x_{ij}}{p_{ij}} \geq 1 \qquad \forall j \tag{17}$$

$$\sum_{r_j \in [t,t']} x_{ij} \leq (t' - t) + D \qquad \forall i, \forall t, t' \in \{r_1, \ldots, r_n\} \tag{18}$$

$$x_{ij} \geq 0 \qquad \forall i, j \tag{19}$$

$$x_{ij} = 0 \qquad \forall i, j \quad \text{with } p_{ij} > D. \tag{20}$$

Remark: Note that the variables x_{ij} do not specify the time at which job j is assigned to machine i. However, it is instructive to view x_{ij} units of work being assigned at time r_j (the release time of j).

We say that a job is *integrally* assigned to machine i in the interval $[t_1, t_2]$ if $x_{ij} = p_{ij}$ and $r_j \in [t_1, t_2]$. Similarly, if $x_{ij} > 0$ and $x_{ij} \neq p_{ij}$, then job is assigned fractionally to machine i. Let $p_{\max}$ denote the maximum value of p_{ij} is some optimum schedule (note that $p_{ij} \leq D$). For convenience of description later, let us also assume that the release times are distinct (say, by perturbing them by some infinitesimally small amount).

As previously, we prove Theorem 1.2 using iterated rounding. To this end, we will show how to create a "tentative" schedule satisfying the following properties.

Lemma 3.1. *There exists a solution $x^* = \{x_{ij}\}_{i,j}$ with the following properties:*

- *x^** integrally assigns *each job j to a single machine i; i.e., x_{ij} is equal to p_{ij} for some machine i.*
- *For any time interval $[t_1, t_2]$, the total volume of jobs assigned in x^* is at most $(t_2 - t_1) + D + O(\log n) \cdot p_{\max}$. That is,*

$$\sum_{j : r_j \in [t_1, t_2]} x_{ij} \leq (t_2 - t_1) + D + O(\log n) \cdot p_{\max}.$$

We first show Theorem 1.2 follows easily from the above lemma.

Proof of Theorem 1.2. Given a solution x^* satisfying the properties of Lemma 3.1, we construct a valid schedule such that flow-time of each job is at most $D + O(\log n) \cdot p_{\max}$ as follows. Fix a machine i. Consider the jobs $J(i, x^*) = \{j \mid x_{ij} = p_{ij}\}$ assigned to machine i, and schedule them in First In First Out (FIFO) order.

To see that every job is completed by time $r_j + D + O(\log n) \cdot p_{\max}$, fix a job j and consider the interval $[0, r_j]$. Let $t' \in [0, r_j]$ be the latest time instant when the machine i is idle. This implies that all the jobs in $J(i, x^*)$ released in the interval $[0, t']$ are completed by t'. As the machine is busy during $(t', r_j]$ and the total volume of jobs assigned in the interval is at most $(r_j - t') + D + O(\log n) \cdot p_{\max}$, the total volume of jobs alive at r_j is at most $D + O(\log n) \cdot p_{\max}$, which implies the result. □

Henceforth we focus on proving Lemma 3.1.

3.1 Iterated Rounding and proof of Lemma 3.1

We prove Lemma 3.1 using iterated rounding. Similar to the proof of Lemma (1.1), we write a successive relaxations of the LP (17-19) denoted by $LP(\ell)$ (21-23), for $\ell = 0, 1, 2...$, such that number of constraints drop by a constant fraction on each iteration. Finally, we obtain a solution where each job is integrally assigned to a single machine. $LP(0)$ is same as LP (17-19). Let $J(\ell)$ denote the set of jobs which are yet to be integrally assigned at the beginning of iteration ℓ. Let $J(0) = J$. We now define $LP(\ell)$ for $\ell \geq 1$.

- *Computing a basic feasible solution:* Solve $LP(\ell-1)$ and find a basic feasible solution $x^*(\ell-1) = \{x_{ij}^{\ell-1}\}_{i,j}$ to $LP(\ell-1)$. We use $x_{ij}^{\ell-1}$ to indicate the value taken by variable x_{ij} in the solution $x^*(\ell-1)$. Initialize $J(\ell) = J(\ell-1)$.

- *Eliminating zero variables:* Variables x_{ij} of $LP(\ell)$ are defined with respect to set of positive variables in the basic feasible solution to $LP(\ell-1)$. In other words, if $x_{ij}^{\ell-1} = 0$ in $x^*(\ell-1)$, then x_{ij} is not defined in $LP(\ell)$.

- *Fixing integral assignments:* If $x_{ij}^{\ell-1} = p_{ij}$ for some job j, then j is permanently assigned to machine i in the solution x^*, and we update $J(\ell) = J(\ell) \setminus \{j\}$.

  We drop all the variables involving job j in $LP(\ell)$, and the constraint (21). Moreover, we update the constraints of type (22) as follows.

- *Defining Intervals:* For each machine i and for each iteration ℓ, we define the notion of intervals $I(i, a, \ell)$ as follows: Consider the variables x_{ij} for jobs $j \in J(\ell)$ (i.e. the ones not assigned integrally thus far), in the order of non-decreasing release times. Greedily group consecutive x_{ij} variables (starting from the beginning) such that sum of the $x_{ij}^{\ell-1}$ values in that group first exceeds $2p_{\max}$. We call these groups intervals, and denote the a-th group by $I(i, a, \ell)$. We say $j \in I(i, a, \ell)$ if $x_{ij} \in I(i, a, \ell)$, and define $\text{Size}(I(i, a, \ell)) = \sum_{j \in I(i,a,\ell)} x_{ij}^{\ell-1}$.

  Note that $\text{Size}(I(i, a, \ell)) \in [2 \cdot p_{\max}, 3 \cdot p_{\max})$ (except possibly for the last interval, in which case we add a dummy job of size $2p_{\max}$.)

LP(ℓ): We are now ready to write $LP(\ell)$.

$$\sum_i \frac{x_{ij}}{p_{ij}} \geq 1 \qquad \forall j \in J(\ell) \tag{21}$$

$$\sum_{j \in I(i,a,\ell)} x_{ij} \leq \text{Size}(I(i, a, \ell)) \qquad \forall i, a, \ell \tag{22}$$

$$x_{ij} \geq 0 \qquad \forall i, j \geq 0 \tag{23}$$

By the definition of intervals and their sizes, it is clear that the feasible solution $x^*(\ell-1)$ to $LP(\ell-1)$ also is a feasible solution to $LP(\ell)$. Next we show that each job is integrally assigned after $O(\log n)$ iterations.

Bounding the number of iterations: Let N_ℓ denote the number of jobs during the ℓ-th iteration.

Lemma 3.2. *For all $\ell > 1$, $N_\ell \leq \frac{N_{\ell-1}}{2}$.*

Proof. Consider the basic optimal solution $x^*(\ell-1)$ to $LP(\ell-1)$. Let $\mathcal{S}_{\ell-1}$ denote the non-zero variables in this solution, i.e. x_{ij} such that $x_{ij}^{\ell-1} > 0$. Consider a linearly independent family of tight constraints in $LP(\ell-1)$ that generate the solution $x^*(\ell-1)$. Since tight constraints of the type of $x_{ij}^{\ell-1} = 0$ only lead to 0 variables, it follows that $|\mathcal{S}_{\ell-1}|$ is at most the number of tight constraints (21) or tight capacity constraints (22). Let $C_{\ell-1}$ denote the number of tight capacity constraints. Thus,

$$|\mathcal{S}_{\ell-1}| \leq N_{\ell-1} + C_{\ell-1} \tag{24}$$

Recall that $A(\ell-1)$ denotes the set of jobs that are assigned integrally in the solution $x^*(\ell-1)$. Then, $N_\ell = N_{\ell-1} - |A(\ell-1)|$ is the number of remaining jobs that are considered in $LP(\ell)$. As each job not in $A(\ell-1)$ contributes at least a value of two to $|\mathcal{S}_{\ell-1}|$, we also have

$$|\mathcal{S}_{\ell-1}| \geq |A(\ell-1)| + 2(N_\ell - |A(\ell-1)|) = N_{\ell-1} + N_\ell \tag{25}$$

Together with (24) this gives

$$N_\ell \leq C_{\ell-1} \tag{26}$$

We now show that $C_{\ell-1} \leq N_{\ell-1}/2$, which together with (26) would imply the claimed result. We know that size of each interval in $(\ell-1)$-th iteration is at least $2 \cdot p_{\max}$. As each tight interval $I(i,a,\ell-1)$ has $\sum_{j \in I(i,a,\ell-1)} x_{ij}^{\ell-1} = \text{Size}(I(i,a,\ell))$, we have

$$N_{\ell-1} \geq \frac{\sum_{i,j} x_{ij}^{\ell-1}}{p_{\max}} \geq \frac{2 \cdot p_{\max} \cdot C_{\ell-1}}{p_{\max}} \geq 2C_{\ell-1}$$

Thus we get $C_{\ell-1} \leq N_{\ell-1}/2$. □

Therefore, number of jobs which are integrally assigned at each iteration ℓ is at least $N_\ell/2$. Now note that number of constraints in $LP(1)$ is at most $n/2$ since size of each interval is at least $2 \cdot p_{\max}$. Hence, the algorithm terminates in $O(\log n)$ rounds.

Bounding the overload: It remains to show that for any time interval $[t_1, t_2]$, the total size of jobs assigned in the interval $[t_1, t_2]$ in x^* is at most $(t_2 - t_1) + O(\log n) \cdot p_{\max} + D$.

Let $\mathsf{Vol}(t_1, t_2, i, \ell)$ be the total volume of jobs assigned (both fractionally and integrally) during the period $[t_1, t_2]$ at the end of ℓ-th iteration. Moreover, let $A(t_1, t_2, i, \ell-1)$ be the set of jobs assigned in the period $[t_1, t_2]$ in the $(\ell-1)$-th iteration, i.e. $x_{ij}^{\ell-1} = p_{ij}$ and $r_j \in [t_1, t_2]$.

Given the solution $x^*(\ell)$ to $LP(\ell)$. Clearly,

$$\mathsf{Vol}(t_1, t_2, i, \ell) = \sum_{r_j \in [t_1, t_2]} x_{ij}^{\ell} + \sum_{\ell' < \ell} \sum_{j \in A(t_1, t_2, i, \ell')} p_{ij}. \tag{27}$$

The following lemma shows that for any time period, the volume does not increase much in each round.

Lemma 3.3. *For any iteration ℓ, machine i, and any time period $[t_1, t_2]$,*

$$\mathit{Vol}(t_1, t_2, i, \ell) \leq \mathit{Vol}(t_1, t_2, i, \ell-1) + 6 \cdot p_{\max}$$

Proof. Consider the maximal contiguous set of intervals $\mathcal{I} = \{I(i,b,\ell), I(i,b+1,\ell), \ldots I(i,b+h,\ell)\}$ such that for every interval $I(i,b',\ell) \in \mathcal{I}$, there exists a job $j \in I(i,b',\ell)$ and $r_j \in [t_1,t_2]$. Recall that size of each interval in $LP(\ell)$ is at most $3 \cdot p_{\max}$. Hence, the intervals $I(i,b,\ell)$ and $I(i,b+h,\ell)$ which overlap $[t_1,t_2]$ at the left and right boundaries respectively, contribute at most $6 \cdot p_{\max}$ to the interval $[t_1,t_2]$. Therefore,

$$\begin{aligned}
\sum_{r_j \in [t_1,t_2]} x_{ij}^{\ell} &\leq \sum_{a=b+1}^{b+h-1} \text{Size}(I(i,a,\ell)) + 6 \cdot p_{\max} \qquad \text{from (22)} \\
&\leq \sum_{r_j \in [t_1,t_2]} x_{ij}^{\ell-1} - \sum_{j \in A(t_1,t_2,i,\ell-1)} p_{ij} + 6 \cdot p_{\max} \qquad \text{from def. of intervals} \\
&\leq \mathsf{Vol}(t_1,t_2,i,\ell-1) - \sum_{\ell' \leq (\ell-1)} \sum_{j \in A(t_1,t_2,i,\ell')} p_{ij} + 6 \cdot p_{\max} \qquad \text{from (27)}
\end{aligned}$$

The lemma now follows by rearranging the terms and using (27). □

Lemma 3.4. *In the solution x^*, the total volume of jobs assigned in any interval $[t_1,t_2]$ is at most $(t_2 - t_1) + D + O(\log n) \cdot p_{\max}$.*

Proof. Consider the interval $[t_1,t_2]$. From the constraints of $LP(0)$ over the interval $[t_1,t_2]$ and definition of $\mathsf{Vol}(i,a,0)$ (equation 27), we have,

$$\mathsf{Vol}(t_1,t_2,i,0) = \sum_{r_j \in [t_1,t_2]} x_{ij}^0 = \quad \leq \quad t_2 - t_1 + D$$

The result now follows by applying Lemma 3.3 for the $O(\log n)$ iterations of the algorithm. □

Proof of Lemma 3.1. From lemma 3.2 we know that each job is integrally assigned to a single machine. Lemma 3.4 guarantees that total volume of jobs assigned in each time interval $[t_1,t_2]$ is bounded by $(t_2 - t_1) + D + O(\log n) \cdot p_{\max}$. This gives us the desired x^* and concludes the proof. □

References

[1] Christoph Ambühl and Monaldo Mastrolilli. On-line scheduling to minimize max flow time: an optimal preemptive algorithm. *Oper. Res. Lett.*, 33(6):597–602, 2005.

[2] S. Anand, Karl Bringmann, Tobias Friedrich, Naveen Garg, and Amit Kumar. Minimizing maximum (weighted) flow-time on related and unrelated machines. In *ICALP (1)*, pages 13–24, 2013.

[3] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241, 2012.

[4] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *In Proc. 15th Symp. on Parallel Algorithms and Architectures (SPAA*, pages 11–18. ACM, 2003.

[5] Baruch Awerbuch, Yossi Azar, Stefano Leonardi, and Oded Regev. Minimizing the flow time without migration. *SIAM J. Comput.*, 31(5):1370–1382, 2002.

[6] Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. *ACM Transactions on Algorithms*, 3(4), 2007.

[7] Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. In *IEE Symposium on the Foundations of Computer Science*, pages 407–414, 2010.

[8] Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In *Symposium on Theory of Computing*, pages 679–684, 2009.

[9] Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *STOC*, pages 84–93, 2001.

[10] Yefim Dinitz, Naveen Garg, and Michel X. Goemans. On the single-source unsplittable flow problem. In *FOCS*, pages 290–299, 1998.

[11] N. Garg and A. Kumar. Better algorithms for minimizing average flow-time on related machines. In *ICALP (1)*, 2006.

[12] Naveen Garg and Amit Kumar. Minimizing average flow time on related machines. In *STOC*, pages 730–738, 2006.

[13] Naveen Garg and Amit Kumar. Minimizing average flow-time : Upper and lower bounds. In *FOCS*, pages 603–613, 2007.

[14] Naveen Garg, Amit Kumar, and V. N. Muralidhara. Minimizing total flow-time: The unrelated case. In *ISAAC*, pages 424–435, 2008.

[15] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42(2):83–97, 2011.

[16] Hans Kellerer, Thomas Tautenhahn, and Gerhard J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *STOC*, pages 418–426, 1996.

[17] Lap-Chi Lau, R. Ravi, and Mohit Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[18] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *J. Comput. Syst. Sci.*, 73(6):875–891, 2007.

[19] Kirk Pruhs, Jiri Sgall, and Eric Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. CRC Press, 2004.

[20] René A. Sitters. Minimizing average flow time on unrelated machines. In *WAOA*, pages 67–77, 2008.